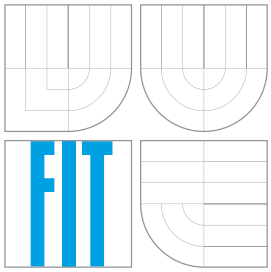
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

RYCHLÝ PRŮSEČÍK PAPRSKU SE SCÉNOU

FAST RAY-SCENE INTERSECTION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MARTIN STŘÍŽ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ADAM HEROUT, Ph.D.

BRNO 2008

Abstrakt

Zobrazení scény metodou sledování paprsku patří k používaným zobrazovacím metodám. Ačkoliv se nejedná o fotorealistickou zobrazovací metodu, poskytuje výstup s vysokou kvalitou obrazu. Její nevýhodou je značná výpočetní náročnost, proto se v praxi používají různé optimalizace. Práce se zabývá optimalizací dělením prostoru, konkrétně pomocí BSP a KD stromů a jejich vzájemným srovnáním.

Klíčová slova

sledování paprsku, dělení prostoru, BSP stromy, KD stromy, optimalizace, 3-D scéna, Phongův osvětlovací model, zákon odrazu, zákon lomu

Abstract

Ray-tracing of 3-D scene is one of the contemporary used rendering methods. Altough it is not a photorealistic method, it produces results with high image quality. Main disadvantage of this method is that it needs a large amount of processing power. That is why various optimalizations must be implemented. This thesis is focused on spatial subdivision optimalizations, namely BSP and KD trees, and their comparsion.

Keywords

raytracing, spatial subdivision, BSP trees, KD trees, optimalization, 3-D scene, Phong's lighting model, law of reflection, law of refraction

Citace

Martin Stříž: Rychlý průsečík paprsku se scénou, bakalářská práce, Brno, FIT VUT v Brně, 2008

Rychlý průsečík paprsku se scénou

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Adama Herouta, Ph.D.

.....
Martin Stříž
3. května 2008

Poděkování

Chtěl bych poděkovat vedoucímu práce Ing. Adamu Heroutovi, Ph.D. a Dr. Ing. Petru Peringerovi za jejich odbornou pomoc při zpracování tohoto tématu.

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2007/2008

Zadání bakalářské práce

Řešitel: **Střiž Martin**
Obor: Informační technologie
Téma: **Rychlý průsečík paprsku se scénou**
Kategorie: Počítačová grafika

Pokyny:

1. Seznamte se s metodami založenými na sledování paprsku a dalšími metody z oblasti počítačové grafiky, v nichž se používá výpočet průsečíku paprsku s komplexní scénou.
2. Vyhledejte, prostudujte a popište metody výpočtu průsečíku paprsku s komplexní prostorovou scénou.
3. Implementujte vybrané metody.
4. Vyhodnoťte efektivitu implementovaných metod výpočtu průsečíku paprsku se scénou, srovnejte implementované řešení s dostupnými alternativami.
5. Znodhoťte dosažené výsledky a navrhněte možnosti pokračování projektu; vytvořte plakátek pro prezentování projektu.

Literatura:

- dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:

- body 1.-2., experimenty směřující k řešení bodu 3.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Herout Adam, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2007
Datum odevzdání: 14. května 2008

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Pavel Zemčík
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Martin Stříž**
Id studenta: 78744
Bytem: Havlíčkovo náměstí 731/16, 708 00 Ostrava
Narozen: 13. 07. 1985, Vítkovice
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Rychlý průsečík paprsku se scénou
Vedoucí/školitel VŠKP: Herout Adam, Ing., Ph.D.
Ústav: Ústav počítačové grafiky a multimédií
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:

tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Obsah

Kapitola 1

Úvod

Zobrazení scény metodou vykreslování paprsku (*raytracing*) patří k používaným zobrazovacím metodám. Poskytuje vysokou kvalitu výstupu, protože dokáže kvalitně modelovat optické jevy odrazu a lomu světla.

Druhá kapitola obsahuje teoretický úvod do raytracingu, popisuje princip jeho činnosti a použité osvětlovací modely, shrnuje výhody a nevýhody.

Hlavním cílem této práce je rychlé zobrazení scény, proto jsem se dále zaměřil na možnosti optimalizace, především na optimalizace dělením prostoru pomocí BSP a KD stromů. Způsob jejich konstrukce, použití a dopad na rychlost vykreslování je popsána v kapitolách 2.4 a 2.5.

Dále se v práci zabývám zvýšením kvality výstupu vykreslování pomocí vyhlazování hran a distribuovaného raytracingu (měkké stíny, rozmazané odrazy). Více v kapitole 2.6. Jedná se o metody velice výpočetně náročné, jejich rychlost je závislá na optimalizacích vykreslování, bez nich není prakticky možné vykreslit scénu v rozumném čase.

Implementace experimentální aplikace pro raytracing je detailně rozebrána v kapitole 3. Důraz je zde kladen na kvalitní objektový návrh, aby bylo možné aplikaci v budoucnu jednoduše rozšířit nebo upravit. Dále zde řeším způsob paralelizace aplikace. V dnešní době dostupných vícejádrových procesorů je vhodné, aby vykreslování mohlo použít všech dostupných prostředků pro zrychlení výpočtů. V této kapitole se také zabývám možnostmi a problémy s portabilitou aplikace.

Poslední kapitola se zabývá testováním výkonu implementované aplikace. Jsou zde srovnávány datové typy s plovoucí řádovou čárkou a jejich vliv na rychlost výpočtu. Hlavní částí kapitoly je porovnání BSP a KD stromů použitých pro optimalizaci vykreslování.

Kapitola 2

Teorie

2.1 Zobrazení metodou sledování paprsku

Zobrazení metodou sledování paprsku (*raytracing*) je jedna z používaných metod pro zobrazení 3D scény. Dosahuje vysoké kvality zobrazení (odraz světla, lom světla, stíny). Základ metody spočívá ve sledování paprsků vysílaných z kamery do scény. Princip můžeme shrnout do následujících bodů:

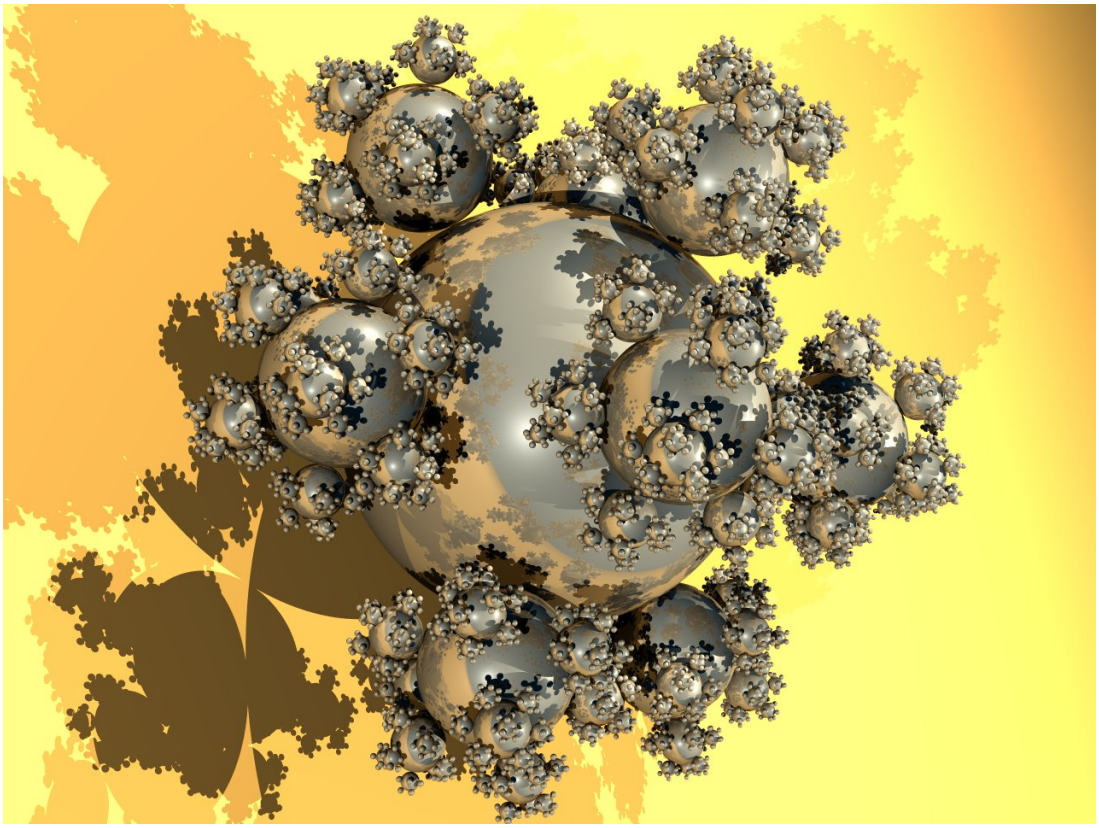
1. Vyšleme paprsek z kamery skrz každý bod průmětny.
2. Najdeme nejbližší objekt ve scéně, který byl paprskem zasažen a vypočítáme průsečík.
3. Od průsečíku vyšleme ke každému světlu ve scéně tzv. stínový paprsek. Pokud během cesty ke světlu paprsek zasáhne objekt, tak světlo nepřispívá k výsledné intenzitě osvětlení. To znamená, že bod leží ve stínu.
4. Pokud světlo přispívá k osvětlení průsečíku, použitím Phongova světelného modelu určíme intenzitu osvětlení [1].
5. Pokud je objekt z materiálu, který odráží světlo, vyšleme podle zákona odrazu rekurzivně nový paprsek. Průhlednými objekty a vysláním paprsku podle zákona lomu světla se v této práci nebudu zabývat. Omezující podmínkou vyslání nového paprsku je maximální hloubka rekurze.
6. Výslednou intenzitu získáme sečtením výsledků předchozích dvou operací.

Může se zdát zvláštní, že trasování probíhá z kamery směrem ke světlu a ne naopak. Důvodem je zvýšení výkonu vykreslování, naprostá většina paprsků vycházejících ze světelných zdrojů nedopadá do našeho oka, proto není potřebná pro výpočet.

Existují metody zobrazení 3D scény jako například *path tracing* [2] nebo *photon mapping*, které trasují paprsky směrem od světelných zdrojů. Tyto metody mají vyšší kvalitu výstupu, protože jsou schopny modelovat optické jevy, které raytracing nedokáže, ale také jsou řádově pomalejší.

2.1.1 Kamera

Prvním krokem algoritmu pro raytracing scény je vyslání paprsku z kamery přes průmětnu. Paprsek je popsán počátkem O a normalizovaným směrovým vektorem r, který získáme z cílového bodu na průmětně $P_{(x,y)}$ pomocí vztahu:



Obrázek 2.1: Testovací scéna s velkým počtem objektů.

$$\vec{r} \quad = \quad \frac{O \ - P_{(x,y)}}{\left|O \ - P_{(x,y)}\right|}$$

Existují dva přístupy, jak realizovat kameru. První, který je jednodušší na implementaci, předpokládá umístění průmětny do roviny XY tak, aby její střed ležel ve středu souřadnicového systému. Počátek (kameru) umístíme na souřadnice $(0, 0, -f)$, vzdálenost f vypočítáme pomocí šířky průmětny w (vodorovné rozlišení obrazu) a úhlu pohledu α:

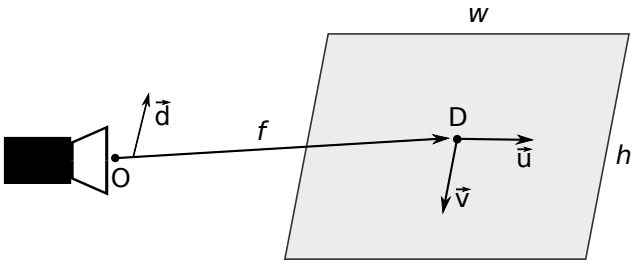
$$f \quad = \quad \left| \frac{w}{2 \operatorname{tg}\left(\frac{\alpha}{2}\right)} \right|$$

Souřadnice bodů na průmětně $P_{(x,y)}$ poté budou odpovídat (h je výška průmětny):

$$P_{(x,y)} \quad = \quad \left(x - \frac{w}{2}, y - \frac{h}{2}, 0 \right)$$

Výhodou uvedeného řešení je rychlost výpočtu směrového vektoru paprsku, nevýhodou je nedostatečná flexibilita. Pokud budeme požadovat zobrazení scény z jiného úhlu nebo jiné vzdálenosti, musíme odpovídajícím způsobem přepočítat souřadnice všech objektů a světel, což je velice nepraktické.

Druhým přístupem je kamera zadaná dvěma body (počátek O a cílem D) a vektorem $\vec{d}$, který určuje natočení kamery (ukazuje „nahoru“). Jedná se o složitější řešení, které ale poskytuje daleko více možností. Grafické znázornění je na obrázku 2.2.



Obrázek 2.2: Kamera zadaná pomocí dvou bodů.

Základem jsou dva vektory $\vec{u}$ a $\vec{v}$. Vektor $\vec{u}$ určuje posun v prostorových souřadnicích, pokud se na průmětně posuneme o jeden pixel na ose X. Vektor $\vec{v}$ určuje posun na ose Y.

$$\begin{aligned} \vec{p} &= f\frac{O - D}{|O - D|} \\ \vec{u} &= \frac{\vec{d} \times \vec{p}}{\left|\vec{d} \times \vec{p}\right|} \\ \vec{v} &= \frac{\vec{p} \times \vec{u}}{|\vec{p} \times \vec{u}|} \end{aligned}$$

Bod na průmětně následně vypočítáme podle vztahu:

$$P_{(x,y)} = O + \vec{p} + \left(x - \frac{w}{2}\right)\vec{u} + \left(y - \frac{h}{2}\right)\vec{v}$$

2.1.2 Výpočet průsečíků s objekty

Pro každý paprsek najdeme nejbližší objekt ve scéně, který byl tímto paprskem zasažen, a vypočítáme průsečík. Nejjednodušším způsobem je otestovat na průsečík všechny objekty ve scéně a následně vybrat nejbližší. S rostoucím počtem paprsků, objektů a světel je však tento postup velmi neefektivní. Pro scény s tisíci objektů je prakticky nepoužitelný. Možnostmi optimalizace se budu zabývat v kapitole 2.3.

V této práci jsem se rozhodl pro podporu dvou základních typů objektů – koule a trojúhelníku. Průsečík s koulí lze zjistit dvěma způsoby:

- Algebraicky – metoda spočívá v dosazení rovnice polopřímky paprsku $\vec{x}(t) = o + t\vec{d}$ do rovnice koule $(x - x_c)^2 + (y - y_c)^2 + (z - z_c)^2 = r^2$. Postup vede na řešení kvadratické rovnice.

- Geometricky – metoda je založena na výpočtu vzdálenosti polopřímky paprsku a středu koule. Pokud je vzdálenost menší než poloměr koule, paprsek kouli protnul.

Oba způsoby jsou detailně popsány v [3]. Rozhodl jsem se pro použití geometrického řešení, protože umožňuje detekovat paprsky neprotínající kouli dříve než algebraické řešení, což ho činí efektivnějším. Pro průsečík s trojúhelníkem také existuje několik algoritmů. Po prozkoumání velkého počtu možností [4] jsem se rozhodl pro algoritmus *Möller-Trumbore* [5], který dosahuje velice dobrých výsledků při nízké paměťové náročnosti.

2.1.3 Osvětlovací modely

V předchozí kapitole jsme zjistili souřadnice průsečíku s nejbližším objektem, který paprsek protnul. Dalším krokem je výpočet intenzity osvětlení. Povrch objektu je definován svými optickými vlastnostmi (drsný, lesklý, zrcadlový). Výpočet realistického osvětlení podle fyzikálního modelu je velice výpočetně náročné, proto se v praxi pro zjednodušení používá empirických modelů.

Lambertův osvětlovací model počítá pouze s difúzním odrazem světla. Jedná se o ideální difuzi, světlo je odraženo do všech směrů se stejnou intenzitou. Velikost intenzity závisí na difúzním koeficientu k_d (materiálová konstanta) kosinu úhlu dopadu světla α.

$$I_l = k_d(\vec{L} \cdot \vec{N})i_d$$

Vektor $\vec{L}$ ukazuje z bodu ke světlu, $\vec{N}$ je normála povrchu a i_d je intenzita světelného zdroje. Oba vektory jsou normalizované, proto platí:

$$\vec{L} \cdot \vec{N} = \cos\alpha$$

Se zvyšujícím se úhlem mezi oběma vektory se bude postupně snižovat intenzita osvětlení bodu.

Phongův osvětlovací model [1] je rozšířením Lambertova modelu. K difúznímu odrazu přidává ještě zrcadlový (spekulární) odraz a rozptýlenou (ambientní) složku světla. Intenzita osvětlení podle Phongova modelu je dána vztahem:

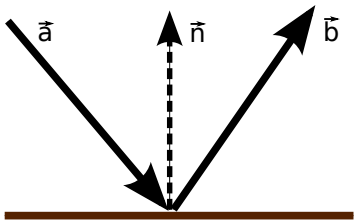
$$\begin{aligned} I_p &= k_a i_a + \sum (k_d(\vec{L} \cdot \vec{N})i_d + k_s(\vec{R} \cdot \vec{V})^n i_s) \\ \vec{R} &= \vec{L} - 2(\vec{L} \cdot \vec{N})\vec{N} \end{aligned}$$

Význam jednotlivých konstant a vektorů je následující:

Symbol	Popis
k_a	Ambientní koeficient materiálu.
k_d	Difúzní koeficient materiálu.
k_s	Spekulární koeficient materiálu.
i_a	Ambientní intenzita světelných zdrojů. Bývá předpočítána jako součet intenzit jednotlivých světel.
i_d	Difúzní intenzita světelného zdroje.
i_s	Spekulární intenzita světelného zdroje.
n	Ostrost odlesku.
$\vec{L}$	Vektor ukazující od bodu ke světelnému zdroji.
$\vec{N}$	Normála povrchu.
$\vec{R}$	Odražený vektor $\vec{L}$ podle normály $\vec{N}$.
$\vec{V}$	Vektor ukazující směrem k pozorovateli.

Zrcadlové povrchy přispívají k intenzitě světla odraženými paprsky. Příchozí paprsek je odražen podle zákona odrazu kolem normály. Názorná ukázka je na obrázku 2.3. Výpočet směrového vektoru odraženého paprsku provedeme dosazením do vzorce

$$\vec{b} = \vec{a} - 2(\vec{a} \cdot \vec{n})\vec{n}$$



Obrázek 2.3: Odraz paprsku podle zákona odrazu.

2.1.4 Výhody a nevýhody raytracingu

Výhodou metody sledování paprsku je realistické zobrazení scény včetně odrazu světla, lomu světla a stínů. Výpočetní nezávislost každého paprsku činí metodu vhodnou pro paralelizaci.

Rychlost zpracování je hlavní nevýhodou. Každý paprsek je počítán zvlášť a mezi výpočty nejsou sdílena žádná data. Vzhledem k rekurzivní povaze trasovacího algoritmu může být počet vyslaných paprsků velmi vysoký.

Další nevýhodou je, že při raytracingu jsou všechna světla bodová (s nulovou velikostí). To má za následek možnost zobrazit pouze ostré stíny, které se v přírodě prakticky nevyskytují. Plošné světelné zdroje lze do jisté míry modelovat; více v kapitole 2.6.2.

2.2 Transformační matice

Pro umisťování a manipulaci s body, vektory a objekty v prostoru lze použít transformační matice.

Nejprve zavedeme pojem *homogenní souřadnice*. Každý bod v prostoru reprezentovaný vektorem (x, y, z) má homogenní souřadnice odpovídající $(x, y, z, 1)$. Reprezentací třírozměrného vektoru ve čtyřrozměrném prostoru nám umožní realizovat potřebné operace posunutí, otočení a změnu měřítka.

Transformaci vektoru $\vec{x}$ pomocí matice A na vektor $\vec{x'}$ provedeme násobením matice vektorem:

$$\vec{x'} = A \cdot \vec{x}$$

Při aplikaci více transformací za sebou postupně násobíme zprava jednotlivými maticemi:

$$\begin{aligned} \vec{x'} &= C \cdot (B \cdot (A \cdot \vec{x})) \\ \vec{x'} &= (C \cdot B \cdot A) \cdot \vec{x} \\ M &= C \cdot B \cdot A \\ \vec{x'} &= M \cdot \vec{x} \end{aligned}$$

Výše uvedený postup ukazuje jednu z výhod maticového přístupu k transformacím: všechny aplikované transformační matice můžeme nejprve vynásobit mezi sebou a až poté aplikovat na bod. To znamená značné snížení počtu vykonaných operací při aplikaci stejných transformací na více bodů.

2.2.1 Transformační matice pro posunutí

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.2.2 Transformační matice pro změnu měřítka

$$S = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.2.3 Transformační matice pro otočení kolem souřadných os

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & 0 \\ 0 & \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad R_y = \begin{bmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_z = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & 0 \\ \sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.3 Optimalizace vykreslování

Vzhledem k tomu, že raytracing je výpočetně velmi náročná metoda, je důležité zaměřit se na její optimalizaci. Jednou z možností optimalizace je použití *adaptivního podvzorkování*. Metoda nevysílá paprsky skrz každý pixel průmětny, ale do mřížky například 8×8 pixelů. Pokud se barva sousedních pixelů v mřížce liší jen nepatrně, je prostor mezi nimi dopočítán interpolací. V opačném případě se pokračuje čtyřmi bloky o velikosti 4×4 pixely atd. Způsob značně urychluje zpracování scén, které mají menší množství detailů [6]. Nevýhodou je snížení kvality obrazu.

Další možností optimalizace je systematicky rozdělit prostor scény tak, aby se při zobrazování minimalizovalo množství počítaných průsečíků s objekty. Způsobů existuje celá řada [7], zde se budu zabývat pouze BSP a KD stromy, protože jsou v současnosti nejvíce používány.

2.4 BSP stromy

BSP (*binary space partitioning*) je metoda, která rekurzivně dělí prostor na dva podprostory pomocí dělící roviny [8]. Výsledkem dělení je datová struktura známá jako BSP strom. Jedná se o binární strom, který obsahuje dva typy uzlů:

- Vnitřní uzel: obsahuje rozměr dělení (X, Y, Z) a pozici dělící roviny
- List: obsahuje seznam objektů scény, které leží v daném podprostoru

2.4.1 Konstrukce stromu

Konstrukce stromu probíhá rekurzivně. Vstupem je osově zarovnaný kvádr ohraničující scénu (*voxel*). Při každém kroku rekurze je rozdělen dělící rovinou na dva menší. Dělící rovina je vybrána tak, aby voxel rozdělila na dvě stejně velké části. Rozměr, ve kterém bude dělení probíhat, může být vybrán postupným střídáním souřadné osy (X, Y, Z, X, ...). Lepší výsledky ale dosáhneme, když jako rozměr dělení použijeme největší rozměr voxelu.

Následně je provedeno rozřazení objektů podle toho, do které poloviny náleží. Pokud objekt protíná dělící rovina, je přiřazen do obou polovin. Rekurzivní dělení je prováděno do té doby, než je splněno jedno ze dvou ukončujících kritérií:

- minimální počet objektů uvnitř voxelu
- maximální hloubka rekurze

Rozřazování je založeno na testu průniku objektu s osově zarovnaným kvádrem. Nejjednodušším způsobem je obalit objekt pomocí kvádru a poté vyřešit, zda kvádry mají průnik nebo ne.

Objekty ale zabírají jen omezenou část kvádru, proto tento přístup detekuje velké množství průniků špatně, což má za následek horší kvalitu vytvořeného stromu. Pro efektivnější řešení je zapotřebí použít speciální algoritmy detekce pro jednotlivé typy objektů.

Test průniku kvádru s koulí pracuje na základě určení nejkratší vzdálenosti kvádru od středu koule. Pokud je vzdálenost větší než poloměr koule, průnik neexistuje. Více o tomto algoritmu naleznete v [9].

Pro průnik kvádru s trojúhelníkem existuje kvalitní algoritmus, který je vysvětlen v publikaci [10].

2.4.2 Průchod stromem

Průchod BSP stromem je realizován rekurzivně. Vstupem procedury průchodu je kořen stromu a voxel ohraničující celou scénu. Při každém kroku rekurze se postupuje následovně:

1. Pokud je uzel listem stromu, otestují se na průsečík všechny jeho objekty. Pokud není průsečík nalezen, paprsek neprotíná žádný objekt ve scéně. V opačném případě je nejbližší z nich nejbližším průsečíkem paprsku s celou scénou.
2. Rozdělíme voxel dělící rovinou na dvě části – přední a zadní.
3. Pokud paprsek protíná pouze jednu z nich, rekurzivně na ni zavoláme funkci průchodu.
4. V případě, že paprsek protíná obě dvě části, projdeme je v pořadí bližší/vzdálenější vzhledem k paprsku.

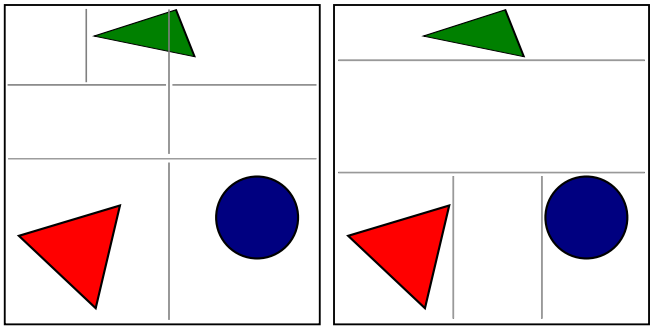
Průchod lze realizovat i bez rekurzivního volání použitím vlastního zásobníku. Podrobně je celý algoritmus včetně pseudokódu popsán v [7].

2.5 KD stromy

KD stromy mají podobnou strukturu jako BSP stromy, liší se pouze v umístění dělící roviny. V případě BSP stromu je umístěna vždy doprostřed děleného voxelu, při použití KD stromu je pozice dělící roviny libovolná.

Vhodná pozice je taková, která vytvoří co největší prázdný prostor, protože ten se ihned v dalším kroku konstrukce stromu stane listem bez objektů. Tím se značně sníží počet testů průsečíků s objekty. Sníží se také průměrná doba průchodu stromu, protože paprsek s větší pravděpodobností protne prázdný voxel.

Pro zjištění optimální pozice dělící roviny využijeme heuristiky SAH, která je popsána v následující kapitole.



Obrázek 2.4: Ilustrační příklad BSP stromu (vlevo) v porovnání s KD stromem (vpravo).

2.5.1 Konstrukce stromu s použitím heuristiky

Heuristika SAH (*surface area heuristic*) je založena na geometrické pravděpodobnosti. Pravděpodobnost jevu, kdy paprsek protne voxel, je závislá na jeho povrchu. Pokud známe pravděpodobnost, můžeme ji použít k výpočtu ceny za průchod voxelem:

$$K = C_T + P \cdot C_I \cdot N$$

C_T a C_I jsou vhodně zvolené konstanty pro cenu za průchod a cenu za průsečík s objektem, N označuje počet objektů a P pravděpodobnost zásahu paprskem.

Při dělení voxelu V na dvě části A a B můžeme cenu vypočítat jako součet cen obou voxelů:

$$\begin{aligned} K &= C_T + C_I \left(P_A N_A + P_B N_B\right) \\ K &= C_T + C_I \left(\frac{S_A}{S_V} N_A + \frac{S_B}{S_V} N_B\right) \end{aligned}$$

Pravděpodobnosti podle SAH odpovídají podílům povrchů dílčích voxelů (S_A a S_B) k voxelu původnímu (S_V). Pro každou možnou pozici dělící roviny vypočítáme cenu za rozdělení K a poté z nich vybereme minimum. Pokud je nalezena minimální cena za rozdělení větší než cena neděleného voxelu

$$K = C_I N_V,$$

nebudeme v dělení pokračovat. Více o cenových modelech a SAH naleznete v pracích [7], [11] a [12].

Protože je počet pozic nekonečný, omezíme výpočty pouze na takové pozice, kde se mění počet objektů N_A a N_B v dílčích voxelech, tzn. na okrajích jednotlivých objektů.

Pokud budeme na každé pozici, která je kandidátem na minimální cenu, počítat objekty nalevo a napravo od dělící roviny, bude mít algoritmus časovou složitost $O(N^2)$. Kromě případů, kdy bude objektů malý počet, je tento přístup nevyhovující.

Hledání minima řazením objektů: Nejprve vytvoříme seznam všech vhodných dělících rovin. U každé z nich si uložíme, kolik objektů v ní má svůj levý okraj (L) a kolik pravý okraj (R). Seznam vzestupně seřadíme a při jeho sekvenčním průchodu upravujeme počty objektů podle následujících pravidel:

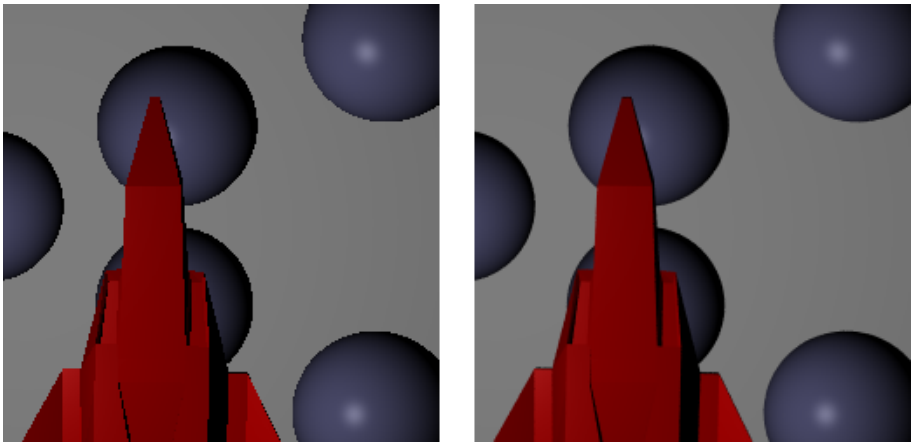
$$\begin{aligned} N_{A_{i+1}} &= N_{A_i} + L_i \\ N_{B_{i+1}} &= N_{B_i} - R_i \end{aligned}$$

Počáteční hodnota N_{A_0} odpovídá počtu objektů, které začínají mimo zpracovávaný voxel, počáteční hodnota $N_{B_0} = 0$. Hledání minima řazením objektů má časovou složitost $O(N^2 \log N)$. Metoda podává velmi dobré výsledky, ale pro velký počet objektů je příliš pomalá.

Hledání minima vzorkováním: Dělící roviny umísťujeme v pravidelných intervalech daných počtem vzorků (zpravidla 8 až 32). Pro každou pozici spočítáme objekty a určíme cenu. Z cen opět zjistíme minimum. Tato metoda je rychlá a podává relativně dobré výsledky při větším počtu objektů, při nižším počtu objektů ale kvalita výsledků klesá.

2.6 Zlepšení vizuální kvality výstupu

2.6.1 Anti-aliasing



Obrázek 2.5: Vlevo výstup bez anti-aliasingu, vpravo s 50 vzorky na pixel.

Výstup vykreslování je vzorkován frekvencí danou výstupním rozlišením. Pokud je rozlišení nízké, může docházet k aliasingu, což se projevuje artefakty v obrazu. „Zubaté“ hrany objektů jsou typickým příkladem.

Jednou z používaných metod anti-aliasingu (vyhlazení hran) je super-sampling, který vzorkuje celý obraz na vyšší frekvenci a následně jej převádí na obraz s frekvencí nižší. V praxi to znamená, že rozdělíme pixel na několik subpixelů a pro každý subpixel vyšleme

samostatný paprsek. Výsledná barva pixelu bude aritmetickým průměrem barev jednotlivých subpixelů.

Super-sampling je velmi náročný na výpočetní výkon, protože se zpracovává daleko větší množství dat. Na druhou stranu podává velmi kvalitní výsledky.

Otázkou zůstává, jak rozdělit pixel na subpixely. Intuitivním přístupem je použití pravidelné mřížky, což je ve většině případů dostačující. V některých případech ale mohou při zpracování vznikat obrazové artefakty. Spolehlivější způsob je použít stochastický super-sampling.

Stochastický super-sampling rozděluje pixel na subpixely pomocí náhodně generovaných souřadnic. Při větším počtu vzorků je pokrytí pixelu téměř rovnoměrné. Algoritmus nevytváří artefakty obrazu jako pravidelná mřížka, často je ale nutné použít větší počet vzorků.

Nejvýhodnější je použít kombinaci pravidelné mřížky a stochastického super-samplingu. Subpixely jsou z pozic v mřížce vychýleny pomocí náhodně generovaných souřadnic. Pro kvalitní výstup je zapotřebí méně vzorků než v případě čistě stochastického přístupu.

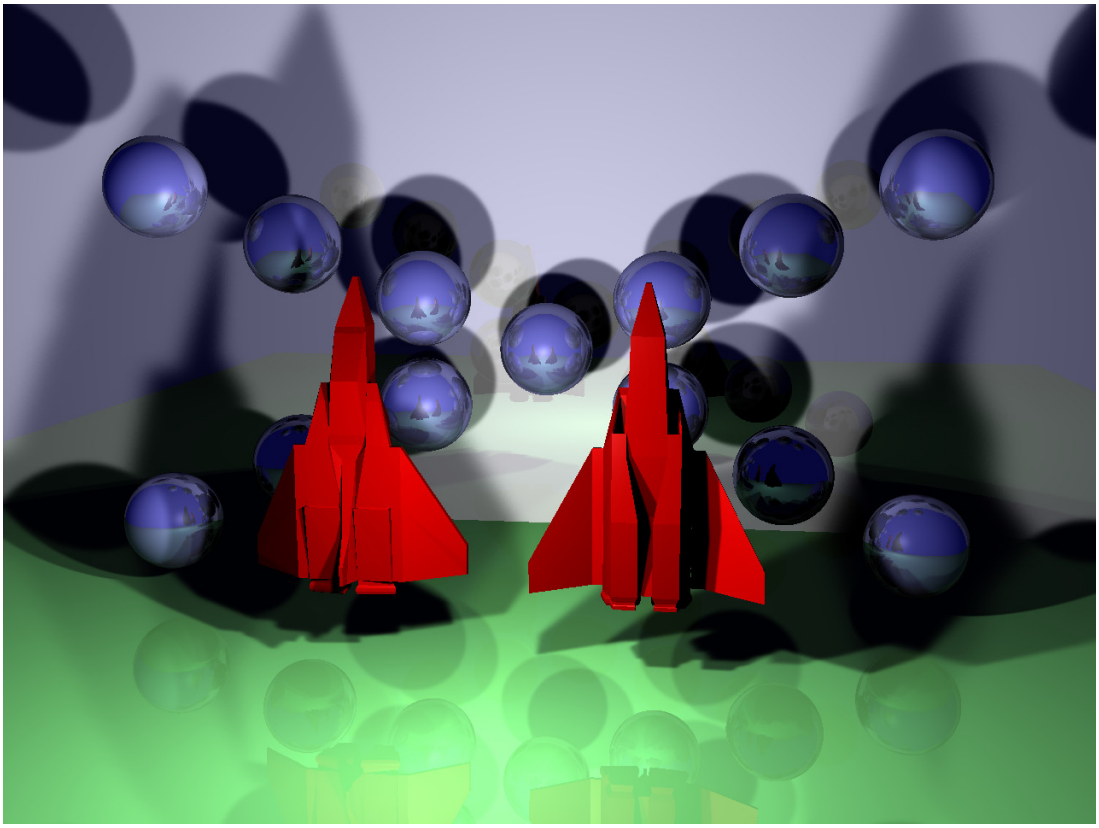
2.6.2 Distribuovaný raytracing

Jednou z nevýhod raytracingu je příliš velká ostrost scény. Všechny vykreslené odrazy a stíny jsou perfektně ostré, což působí poněkud nerealisticky.

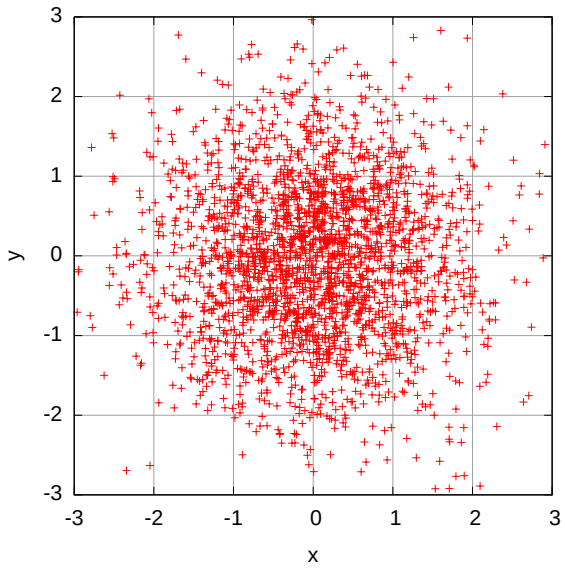
Rozmazaných odrazů simulujících objekt z drsného materiálu dosáhneme pomocí vzorkování odražených paprsků. Namísto jednoho paprsku vygenerujeme při odrazu navíc svazek paprsků distribuovaný okolo původního směru o určitý úhel daný materiálovou konstantou. Jako distribuční funkci je vhodné použít dvourozměrné normální rozdělení (viz obrázek 2.7). Výsledná intenzita je rovna průměru intenzit všech vyslaných paprsků. Jedná se vlastně o přibližné řešení integrálu metodou *Monte Carlo*.

Obdobným způsobem lze řešit měkké stíny. Místo jednoho stínového paprsku vyslaného ke každému světlu opět použijeme distribuční funkci. Příklad je na obrázku 2.6. Více najdete v [2] a [13].

Výhodou distribuovaného raytracingu je realističtější výstup vykreslování, nevýhodou je drastické snížení rychlost výpočtu, protože vytváří velký počet nových paprsků.



Obrázek 2.6: Měkké stíny vytvořené pomocí distribuovaného raytracingu (1000 vzorků na světlo; čas vykreslování 70 minut).



Obrázek 2.7: Příklad normálního rozložení pro 2500 vzorků s rozptylem $\sigma^2 = 1$.

Kapitola 3

Vlastní implementace raytraceru

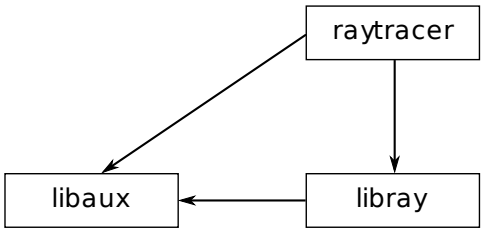
V této kapitole se budu zabývat návrhem a implementací vlastního raytraceru. Kvalitní návrh je základem úspěšné aplikace. Architekturu raytraceru jsem navrhoval poměrně dlouhou dobu, práce se ale nakonec vyplatila: během následné implementace jsem nemusel provádět žádné radikální změny. Značně jsem si tím zkrátil čas vývoje.

3.1 Návrh

První akcí při návrhu byl výběr programovacího jazyka. Podle zadání byla možná implementace v jazyce C nebo v jazyce C++. Zvolil jsem si jazyk C++, protože aplikace je velmi vhodná pro objektové zpracování. Během tvorby objektového návrhu jsem se soustředil především na rozšiřitelnost, čitelnost a portabilitu zdrojového kódu. Dosažení maximálního výkonu pomocí různých optimalizací bylo až mým sekundárním cílem.

Aplikaci jsem rozdělil na tři hlavní komponenty (program a dvě statické knihovny), které jsou mezi sebou navzájem provázány:

- Knihovna `libaux` obsahující zdrojový kód třetích stran jako například xml parser nebo generátor náhodných čísel.
- Knihovna `libray`, která obsahuje vše, co se týká zpracování scény a výpočtů při vykreslování.
- Program `raytracer`, který řeší vstupy a výstupy programu.



Obrázek 3.1: Hlavní komponenty aplikace a jejich závislosti.

3.2 Knihovna libaux

Knihovna obsahuje použitý zdrojový kód třetích stran. Největší součástí je xml parser *TinyXml* [14], který program používá k načtení scény. Jedná se o jednoduše použitelný a kompaktní parser, použil jsem jej hlavně kvůli dřívějším dobrým zkušenostem.

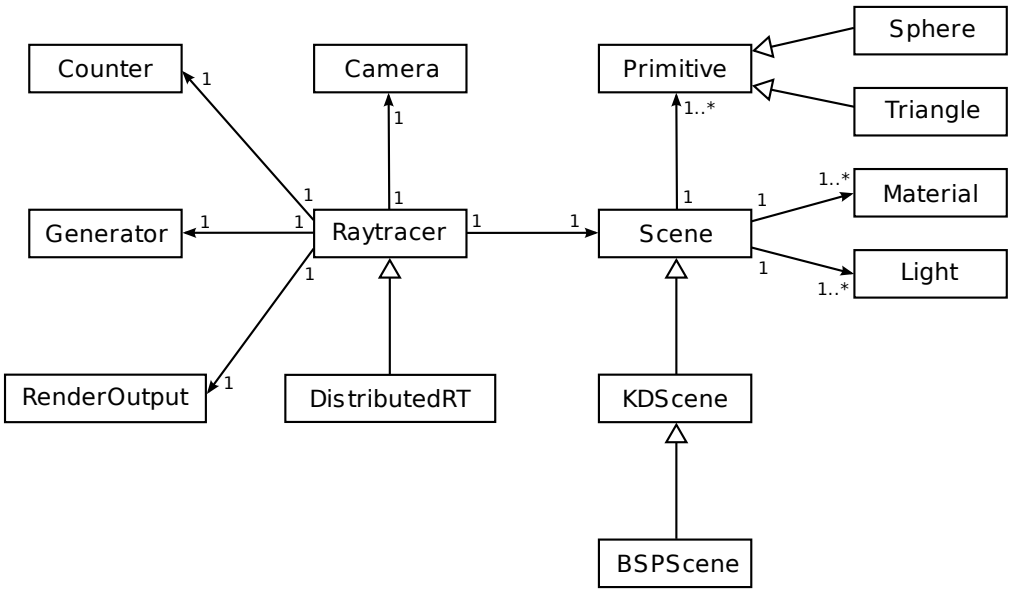
Další součástí je generátor náhodných čísel *Mersenne-Twister* [15]. Tento generátor je velmi rychlý a podává lepší výsledky než standardní C funkce `rand()`. Hodí se především pro generování velkého počtu dat nebo vícerozměrných dat.

Poslední součástí knihovny je funkce pro zjištění průniku trojúhelníku s osově zarovnaným kvádrem [10].

Kód jednotlivých součástí mohl být bez problémů umístěn v ostatních dvou komponentách, ale tímto způsobem jsem jednoznačně oddělil mnou napsaný kód od ostatního.

3.3 Knihovna libray

Hlavním úkolem knihovny je vykreslení zadané scény do renderovacího výstupu. Na obrázku 3.2 můžete vidět diagram tříd, které knihovna používá ke splnění tohoto úkolu (pro zjednodušení jsou v diagramu uvedeny pouze nejdůležitější).



Obrázek 3.2: Diagram tříd knihovny libray.

3.3.1 Datové typy

Většina výpočtů při vykreslování probíhá s čísly s plovoucí řádovou čárkou. Již během návrhu jsem počítal s podporou datových typů `float` a `double` tak, aby se podmíněnou kompilací dal zvolit jeden z nich pro všechny výpočty v aplikaci. Docílil jsem toho zavedením datového typu `real` (stejně jako v ukázkovém ray-traceru [16], který mi byl inspirací), který je aliasem buď na `float` nebo `double`. Na vybraném datovém typu jsou závislé použité

funkce matematické knihovny. Téměř všechny existují ve verzích pro oba datové typy, proto v aplikaci používám vlastní definice preprocesoru, které vždy odkazují na správnou z nich:

```
#ifdef USE_FLOAT
  typedef float real;
  #define real_sqrt sqrtf
  #define real_log logf
#else
  typedef double real;
  #define real_sqrt sqrt
  #define real_log log
#endif
```

Pro vektorové operace jsem vytvořil strukturu `Vector3d`, která představuje vektor v prostoru (x, y, z) nebo barvu (r, g, b). Díky možnosti přetěžování operátorů v jazyce C++ je možné s vektorem pracovat podobně jako s běžným číslem. Navíc jsou implementovány operace pro skalární součin, vektorový součin a normalizaci.

3.3.2 Materiály

Materiál povrchu je charakteristický pro každý objekt umístěný ve scéně, ovlivňuje výpočty osvětlovacího modelu. Natavením jeho vlastností definujeme, jak povrch objektu vypadá. Dostupné vlastnosti jsou:

- Barva povrchu reprezentovaná vektorem (r, g, b). V osvětlovacím modelu se počítá s každou složkou zvlášť.
- Ambientní koeficient materiálu.
- Difúzní koeficient materiálu.
- Spekulární koeficient materiálu.
- Ostrost spekulárního odlesku.
- Odrazivost materiálu.
- Rozptyl difúzního odrazu udává rozptyl normálního rozložení použitého při distribuovaném raytracingu, který byl popsán v kapitole 2.6.2.

3.3.3 Scéna

Jádrem knihovny je scéna. Obsahuje v sobě všechny objekty k vykreslení spolu s použitými materiály a světly. Pro uchování dat je použita standardní C++ třída `std::vector`.

Jak již bylo řečeno v kapitole 2.1.2, podporovanými objekty jsou koule a trojúhelníky. Oba dva typy objektů zděděny z bázové třídy `Primitive`. která poskytuje rozhraní pro hledání průsečíku a získání normály. Proto můžeme jednoduše přidat další typ podporovaného objektu, aniž bychom museli měnit cokoliv v algoritmech zpracování scény.

Hlavním úkolem scény je hledání průsečíků paprsku s objektem. V případě paprsků z kamery nebo paprsků odražených objekty je hledán vždy nejbližší průsečík, v případě stínových paprsků, které slouží ke zjištění, zda je bod osvětlen, je hledán jakýkoliv průsečík do zadané vzdálenosti.

Naivním přístupem bez použití optimalizací je testování vždy všech objektů scény na přítomnost průsečíku. Při velkém množství objektů může být čas vykreslení scény naivním způsobem v řádu hodin nebo dní.

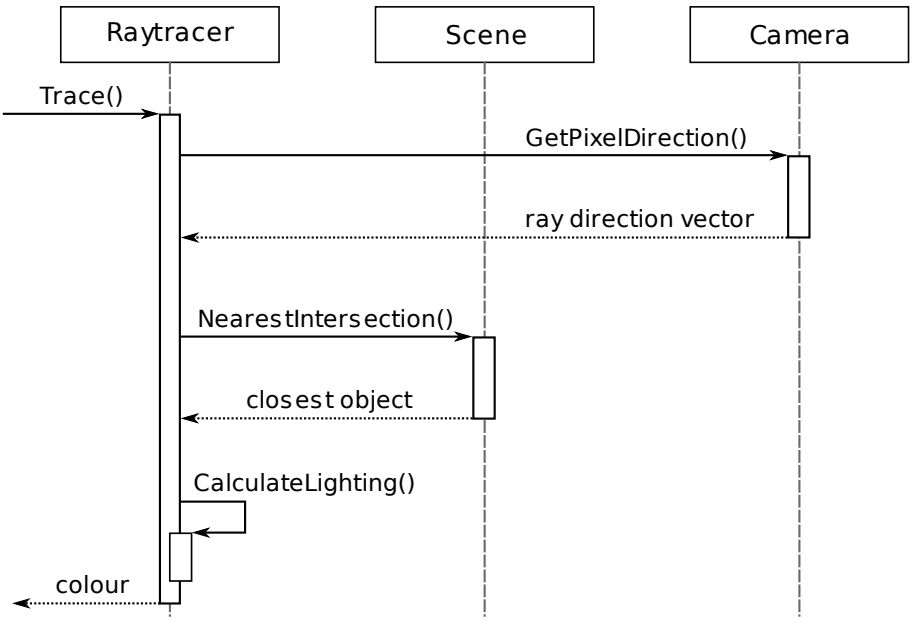
Použitím BSP nebo KD stromů se čas vykreslení výrazně zkrátí. Implementace ve třídách `BSPScene` a `KDScene` obsahují metody pro vytváření a rychlé procházení stromovou datovou strukturou. Během procházení je nutné řešit průsečíky paprsku s osově orientovaným kvádrem. Použitá metoda je popsána v [17]. Pro řídící objekt, který je popsán v následující kapitole, je použití těchto tříd transparentní.

3.3.4 Řízení vykreslování

Řízení vykreslování je prováděno v třídě `Raytracer`. Jejím vstupem je nastavená kamera a scéna obsahující objekty. Kamera se řídí principy popsanými v kapitole 2.1.1, tzn. má nastaven zdrojový a cílový bod, vzdálenost průmětny od počátku a rozlišení obrazu.

Během vykreslování jsou postupně vypočítány barvy bodů (pixelů) průmětny: pro každý bod je nejprve zjištěn směrový vektor paprsku. Následně je vyslán paprsek z kamery do scény a zjištěn průsečík s nejbližším objektem ve scéně. Na základě průsečíku je vypočítána intenzita osvětlení (barva) daného bodu. V případě výskytu reflexního povrchu objektu jsou vyslány další paprsky, které přispívají k výsledné barvě pixelu.

Vypočítaná barva je nakonec zapsána přes rozhraní `RenderOutput` do vykreslovacího výstupu. Celá operace je znázorněna v sekvenčním diagramu na obrázku 3.3.



Obrázek 3.3: Sekvenční diagram vykreslení jednoho pixelu.

V prvních verzích aplikace byly pixely vykreslovány sekvenčně po řádcích. Později jsem algoritmus změnil tak, aby pixely vykresloval po blocích. Ve většině případů to přineslo zvýšení výkonu kvůli tomu, že body blízko sebe sdílí daleko více dat, využije se tak lépe vyrovnávací paměť procesoru. Jako optimální se nakonec ukázala velikost bloku 32×32 pixelů, větší bloky již nepřináší přílišné zlepšení.

Distribuovaný raytracing je implementován třídou `DistributedRT` zděděnou ze třídy `Raytracer`. Obsahuje upravené metody pro výpočet odrazů a stínů, které pracují na bázi vysílání množství paprsků podle distribuční funkce. Generování náhodných čísel s dvourozměrným normálním rozdělením je řešen pomocí Box-Mullerovy transformace [18]. Více o distribuovaném raytracingu naleznete v kapitole 2.6.2.

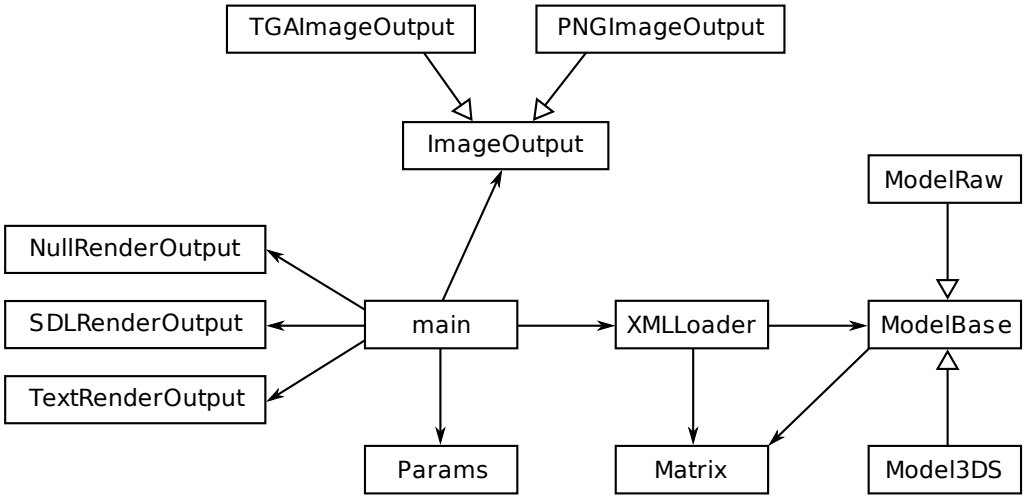
3.3.5 Pomocné třídy

Během vykreslování se používá několik pomocných tříd, první z nich je `Counter`, která zapouzdřuje jednoduchý čítač. Během vývoje byla upravena pro paralelní zpracování (viz kapitola 3.5). Při výpočtech je využita pro počítání vyslaných paprsků (normálních a stínových). Po ukončení výpočtů jsou z ní odečteny výsledky za účelem vypsání statistiky.

Třída `Generator` tvoří rozhraní nad generátorem náhodných čísel z knihovny `libaux` (kapitola 3.2). Také byla upravena během vývoje kvůli paralelního zpracování.

`ProgressUpdater` je rozhraní, které slouží k oznámení o průběhu výpočtu, konkrétně při dokončení vykreslování jednoho bloku. Umožňuje v programu implementovat ukazatel průběhu.

3.4 Program raytracer



Obrázek 3.4: Diagram tříd programu raytracer. Všechny asociace jsou typu 1:1.

3.4.1 Zpracování parametrů

Raytracer je konzolová aplikace, ovládá se z příkazové řádky pomocí parametrů. Po spuštění se všechny parametry příkazové řádky předají třídě `Params`, která provede jejich zpracování. Pokud při zpracování dojde k chybě, je vypsána nápověda k použití a ukončen program.

V opačném případě jsou proměnné objektu třídy `Params` naplněny daty, která jsou pak předána vykreslovací funkci.

3.4.2 Načítání scény

Scéna je načítána ze souboru formátu XML, který je zadán jako parametr příkazové řádky. Veškerá práce se souborem je zapouzdřena ve třídě `XMLLoader`. Příkladem souboru se scénou může být:

```
1  <?xml version="1.0" ?>
2  <scene>
3    <background r="0.1" g="0.1" b="0.6" />
4
5    <camera fov="50" zoom="100">
6      <origin x="4" y="-4.5" z="-18" />
7      <destination x="1" y="-0.5" z="5" />
8    </camera>
9
10   <materials>
11     <material name="blue" r="0.7" g="0.7" b="1.0"
12       diffuse="0.4" reflection="0.4" />
13     <material name="red_steel" r="1" g="0" b="0"
14       diffuse="0.6" specular="0.3" />
15   </materials>
16
17   <models>
18     <model name="spaceship" file="scenes/spaceship.3ds" type="3ds" />
19   </models>
20
21   <objects>
22     <sphere x="0" y="-1" z="12" r="1.2" material="blue" />
23     <scale factor="0.025">
24       <rotate ry="1">
25         <translate x="-3" z="4.5">
26           <model name="spaceship" material="red_steel" />
27         </translate>
28       </rotate>
29     </scale>
30   </objects>
31
32   <lights>
33     <light x="-1" y="0" z="0" r="0.8" g="0.8" b="0.8" size="0.1" />
34   </lights>
35 </scene>
```

Prvním řádkem vždy musí být XML hlavička, poté následuje kořenový element `scene`, ve kterém jsou obsaženy všechny další definice.

Element `background` slouží k nastavení barvy pozadí, pokud není přítomen, je pozadí scény nastaveno na černou barvu. Řádky 5–8 obsahují definici vlastností kamery elementem

`camera`. Kamera bude umístěna do bodu `origin` a nasměrována na bod `destination`. V případě, že není zadán atribut s úhlem pohledu `fov` nebo ohniskovou vzdáleností `f`, je ohnisková vzdálenost nastavena na vzdálenost těchto dvou bodů.

Dále jsou v ukázkovém souboru definovány materiály povrchů. Pomocí atributů lze nastavit všechny aspekty materiálů, které byly popsány v kapitole 3.3.2. Každý materiál je jednoznačně identifikován svým jménem v atributu `name`.

Definice použitých modelů jsou umístěny v elementu `models`. Podle typu je model načten pod zadaným jménem do paměti tak, aby mohly být jeho instance opakovaně vloženy do scény.

Hlavním elementem souboru se scénou je `objects` (řádky 21–30), který řídí vkládání objektů do scény. Může obsahovat dva typy vnořených elementů. Prvním z nich jsou operace, které upravují transformační matici (viz kapitola 2.2):

- `transform` – Posunutí o zadané souřadnice.
- `rotate` – Rotace kolem počátku souřadnicové soustavy o zadaný úhel (atributy `x`, `y`, `z`) nebo násobek π (atributy `rx`, `ry`, `rz`).
- `scale` – Zvětšení, zmenšení nebo převrácení podle zadaných koeficientů.

Transformační elementy lze do sebe libovolně vnořovat a tím transformace skládat. Druhým typem jsou definice objektů:

- `sphere` – Koule zadaná svým středem a poloměrem.
- `triangle` – Trojúhelník zadaný pomocí tří vrcholů.
- `quad` – Čtyřúhelník, který je přidán jako dva trojúhelníky, protože vykreslovací jádro čtyřúhelníky nepodporuje.
- `model` – Vložení instance modelu podle zadaného jména.

Na každý objekt je před vložením aplikována aktuální transformační matice. V ukázkovém příkladu je vložen model s názvem „ships“, který je nejprve zmenšen ve všech osách koeficientem $0,025$, poté otočen kolem osy y o úhel π radiánů a nakonec posunut o -3 jednotky na ose x a $4,5$ na ose z.

Posledním elementem ukázkového příkladu je `lights`, který obsahuje seznam světel scény. Každé světlo je definováno svou pozicí a barvou, případně velikostí, pokud se jedná o plošný světelný zdroj. Velikosti světel jsou zohledněny pouze při použití distribuovaného raytracingu, jinak se všechna světla chovají jako bodová.

3.4.3 Načítání a práce s 3D modely

Jak bylo naznačeno v minulé kapitole, aplikace podporuje načítání modelů, které pak mohou být v několika instancích vloženy do scény. Výhodou použití modelu je přehlednost a zrychlení zpracování XML souboru scény. Aplikace podporuje dva formáty modelů:

- `3ds` (třída `Model3DS`) – jedná se o standardní modely exportované programem 3D Studio MAX. Podporovány jsou pouze soubory obsahující jeden trojúhelníkový model. Kromě něho se ze souboru žádná další data (materiály, osvětlení, ...) nenačítají.
- `raw` (třída `ModelRaw`) – vlastní binární formát pro reprezentaci modelu složeného z trojúhelníků nebo koulí. Kompletní popis formátu je v příloze D.

3.4.4 Výstup vykreslování

Třídy sloužící jako výstup vykreslování jsou zděděny z čistě abstraktní třídy `RenderOutput`. V programu je jich implementováno několik, v případě nutnosti je možné jednoduše doimplementovat další.

`NullRenderOutput` slouží k vykreslování naprázdno, metody třídy neobsahují žádný kód. Tento výstup je vhodný především pro měření výkonu, protože výsledky nejsou zkresleny zápisem do renderovacího výstupu.

`SDLRenderOutput` využívá multiplatformní knihovny SDL (*Simple DirectMedia Layer* k vytvoření okna, ve kterém je poté zobrazen výstup vykreslování.

`ImageRenderOutput` je abstraktní třída pro zápis výstupu do souboru. Ze začátku vývoje z ní byla odvozena pouze třída `TGAImageOutput`, která zapisuje to grafického formátu `Targa`. Formát jsem zvolil, protože je univerzální a jednoduchý na implementaci. Později jsem přidal ještě další třídu `PNGImageOutput` využívající knihovnu libpng pro zápis do souborů formátu PNG (*Portable Network Graphics*).

`TextRenderOutput` vypisuje výstup do textového souboru nebo konzole. Barevný výstup je nejprve převeden na odstíny šedi, které jsou následně převedeny na textovou reprezentaci.

3.4.5 Měření času a ukazatel průběhu

Jedním z cílů práce je srovnávání výkonu raytraceru podle různých kritérií. Abychom mohli měřit výkon, musí být v programu implementován mechanismus pro měření času. V programu je realizován třídou `Stopwatch`, která měří dva časy: procesorový a reálný.

Společně s implementací distribuovaného raytracingu jsem do aplikace přidal třídu `ConsoleProgressBar`, která implementuje rozhraní `ProgressUpdater` a slouží k zobrazení průběhu vykreslování na standardní výstup. Některé scény mohou být zpracovávány až několik hodin, proto je dobré, když uživatel vidí, že aplikace stále pracuje.

3.5 Paralelizace vykreslování

Vykreslování scény je výpočetně velmi náročná operace, kterou je možno snadno paralelizovat, protože jednotlivé paprsky jsou na sobě nezávislé.

Při prvních pokusech implementace paralelního zpracování jsem používal standardní POSIXová vlákna (*pthreads*). Aplikace fungovala spolehlivě, ale zdrojový kód byl složitý a hůře čitelný. Další byl problém s podmíněnou kompilací, chtěl jsem mít možnost, jak jednoduše vytvořit verzi programu, která nepoužívá vlákna. Při použití *pthreads* se čitelnost kódu ještě snížila.

Později jsem narazil na řešení *OpenMP* [19], které umožňuje paralelizovat části programu pouze pomocí direktiv preprocesoru překladače. Příkladem může být kód pro renderování bloků obrazu:

```
1 #ifdef _OPENMP
2 #pragma omp parallel for shared(blocks, ray) private(i) schedule(dynamic)
3 #endif
4 for (i = 0; i < m_blocks_total; i++) {
5   RenderBlock(&blocks[i], ray);
6 }
```

Bloky jsou paralelně zpracovávány všemi dostupnými procesory, knihovna automaticky vytváří počet vláken podle počtu procesorů. Cyklus je rozdělován dynamicky: pokud existuje část cyklu, která není zpracována, dostane ji přiděleno volné vlákno. Podmíněná kompilace není problémem, protože překladač při použití knihovny definuje automaticky symbol `_OPENMP`. Podpora *OpenMP* je implementována například v překladači GCC od verze 4.2. Více informací o možnostech překladu programu najdete v příloze A.

Pokud je program prováděn paralelně, musíme zajistit, aby dvě vlákna nezapisovala současně do stejného paměťového prostoru. Při renderování bloků se problém neprojeví, protože každý blok má svůj vlastní úsek paměti pro pixelů obrazu na výstup. Kritické sekce jsou zapouzdřeny do tříd `Counter` a `Generator`, které jsem musel upravit tak, aby uchovávaly separátní hodnoty pro jednotlivá vlákna.

3.6 Portabilita zdrojového kódu

Aplikace byla primárně vyvíjena na platformě GNU/Linux a překladači GCC verze 4.2. Překladový systém byl postaven na utilitě *GNU make*, jeho popis najdete v příloze A. Zdrojový kód je psán podle normy C++ 98, proto by měl být přeložitelný většinou dnes dostupných překladačů.

Aplikaci se mi podařilo úspěšně přeložit a otestovat na následujících platformách:

HW platforma	**Operační systém**	**Překladač**
x86_64	GNU/Linux	GCC 4.2.3
x86	GNU/Linux	GCC 4.2.3
x86	GNU/Linux	GCC 4.3.0
x86	GNU/Linux	Intel C++ Compiler 10.0.023
x86	FreeBSD	GCC 3.4.6
x86	MS Windows	GCC 3.4.4 (MinGW)
x86	MS Windows	GCC 3.4.4 (Cygwin)

Tabulka 3.1: Testované HW platformy a operační systémy.

Jistou překážkou v portabilitě do systému Microsoft Windows je závislost na standardech *POSIX*, konkrétně ve zpracování parametrů (třída `Params`) a měření času (třída `Stopwatch`). Proto není možné přeložit aplikaci například v Microsoft Visual C++. Problém je možné vyřešit přepsáním uvedených částí do Windows API.

Kapitola 4

Testování aplikace

4.1 Testovací sestavy a metodika měření

Testy probíhaly v operačním systému GNU/Linux, program byl kompilován překladačem GCC 4.2 bez podpory SDL, PNG a OpenMP. Hardwarová konfigurace testovacích stanic byla následující:

- **S1**: Intel Core2 Duo E6420 (64-bit), 2.13 GHz, 4 MB cache, 2048 MB RAM.
- **S2**: AMD Opteron 2216 (64-bit), 2.40 GHz, 1 MB cache, 4096 MB RAM.
- **S3**: AMD Opteron 2216 (32-bit), 2.40 GHz, 1 MB cache, 4096 MB RAM.
- **S4**: Intel Pentium 4 (32-bit), 1.50 GHz, 256 kB cache, 512 MB RAM.
- **S5**: Intel Core Duo T2400 (32-bit), 1.83 GHz, 2 MB cache, 1536 MB RAM.

4.2 Testované scény

Scéna	Trojúhelníky	Koule	Světla
balls	2	7381	3
balls2	2	597871	3
ships	1004	13	2
tree	180182	0	7
pyramid5	18750	0	1

Tabulka 4.1: Charakteristiky testovacích scén.

4.3 Měření výkonu BSP a KD stromů

Srovnání BSP a KD stromů jsem provedl na všech testovacích stanicích. Hlavním měřeným údajem byl procesorový čas. Vykreslování bylo prováděno bez výstupu s rozlišením 800×600 pixelů, zapnutými stíny a odrazy do třetí úrovně rekurze. Kvůli eliminaci chyb bylo každé měření provedeno desetkrát, výsledky jsou střední hodnotou všech měření.

		balls		balls2		ships		tree		pyramid5	
		BSP	KD	BSP	KD	BSP	KD	BSP	KD	BSP	KD
S1	konstrukce	0.40	0.29	0.01	0.09	0.01	0.01	0.96	2.63	0.69	4.73
	vykreslení	0.41	0.40	3.90	2.90	2.40	1.90	148.95	2.75	55.29	6.87
	celkem	0.82	0.69	3.90	2.99	2.40	1.91	149.92	5.37	55.98	11.59
S2	konstrukce	0.43	0.76	0.01	0.23	0.01	0.04	3.15	9.05	1.70	14.61
	vykreslení	0.37	0.95	4.59	3.54	2.82	3.06	621.47	4.51	146.86	12.50
	celkem	0.80	1.71	4.60	3.77	2.83	3.10	624.62	13.56	148.56	27.11
S3	konstrukce	0.84	0.77	0.02	0.20	0.02	0.03	2.25	9.99	1.86	14.71
	vykreslení	0.86	0.79	6.82	4.89	4.90	3.21	658.90	5.23	156.37	12.93
	celkem	1.71	1.56	6.84	5.09	4.91	3.24	661.15	15.22	158.23	27.64
S4	konstrukce	1.37	1.67	0.04	0.36	0.04	0.07	4.34	18.57	2.71	23.72
	vykreslení	1.63	1.75	17.07	13.68	10.70	8.76	608.49	13.57	322.62	34.94
	celkem	3.00	3.42	17.11	14.04	10.74	8.83	612.83	32.14	325.33	58.66
S5	konstrukce	0.60	0.62	0.01	0.16	0.01	0.02	1.73	7.17	1.16	10.31
	vykreslení	0.62	0.69	6.72	5.06	4.30	3.22	330.19	5.04	124.00	12.14
	celkem	1.22	1.31	6.73	5.22	4.31	3.24	331.92	12.21	125.16	22.45

Tabulka 4.2: Časy vykreslení scény s použitím BSP a KD stromů.

Z naměřených hodnot uvedených v tabulce 4.2 je vidět, že ve většině případů je použití KD stromů výhodnější. Delší čas konstrukce stromu oproti BSP je vyvážena někdy až několikanásobně kratším časem vykreslování.

Výjimkou jsou scény s nízkým počtem objektů a scény, které mají objekty rozmístěny tak, že jsou vytvořené stromy velmi podobné. V tomto případě je doba vykreslování prakticky shodná, omezujícím faktorem je pouze doba konstrukce.

Naopak při scénách s nerovnoměrně rozmístěnými objekty (například *tree*) jsou BSP stromy velmi neefektivním řešením, protože algoritmus vytváření brzy dosáhne maximální hloubky rekurze. To má za následek vysoký počet objektů v listech stromu, proto musí aplikace testovat daleko více průsečíků s objekty.

Pro srovnání jsou počty uzlů a průměrné počty testovaných průsečíků na paprsek uvedeny v tabulce 4.1.

Scéna	**BSP uzly**	**KD uzly**	**BSP pr./paprsek**	**KD pr./paprsek**
pyramid5	136363	25391	5.59	10.91
balls	3951	12423	27.46	10.19
ships	1497	1785	11.04	8.67
tree	435	7699	1378.85	21.48
balls2	4379	100157	1101.21	50.37

Tabulka 4.3: Srovnání kvality vytvořených stromů.

4.4 Srovnání datových typů double a float

Jedním z cílů této práce je srovnání použití datových typů pro čísla s plovoucí desetinnou čárkou – `double` (dvojitá přesnost, 64 bitů) a `float` (jednoduchá přesnost, 32 bitů). Jak jsem již naznačil v kapitole 3.3.1, aplikace je uzpůsobena tak, aby bylo možné jednoduše použít jeden nebo druhý datový typ pro všechny výpočty.

V tabulce 4.4 jsou uvedeny naměřené hodnoty pro oba datové typy. Výsledky jsou velmi překvapivé. Na první pohled by se mohlo zdát, že práce s 32-bitovými čísly s jednoduchou

		balls		balls2		ships		tree		pyramid5	
		D	F	D	F	D	F	D	F	D	F
S1	konstrukce	0.29	0.38	0.08	0.08	0.02	0.02	2.67	2.88	4.78	3.93
	vykreslení	0.40	0.55	2.75	3.40	2.30	2.91	2.83	3.50	7.07	8.27
	celkem	0.69	0.93	2.83	3.49	2.32	2.93	5.50	6.37	11.85	12.20
S2	konstrukce	0.44	0.51	0.24	0.23	0.04	0.02	8.43	7.33	13.82	11.41
	vykreslení	0.39	0.59	3.27	3.97	2.59	3.65	3.87	5.10	12.09	11.71
	celkem	0.83	1.10	3.51	4.20	2.63	3.67	12.30	12.43	25.91	23.12
S3	konstrukce	0.63	1.00	0.14	0.22	0.02	0.04	10.09	8.24	13.55	9.65
	vykreslení	0.69	0.69	4.63	4.38	3.07	3.35	5.28	3.63	12.48	9.84
	celkem	1.32	1.69	4.77	4.60	3.09	3.39	15.37	11.87	26.03	19.49
S4	konstrukce	1.65	1.31	0.36	0.32	0.07	0.07	18.46	11.43	23.61	15.21
	vykreslení	1.67	1.23	13.60	9.95	8.85	5.93	13.47	8.46	34.08	25.55
	celkem	3.32	2.54	13.96	10.27	8.92	6.00	31.93	19.89	57.69	40.76
S5	konstrukce	0.62	0.58	0.15	0.16	0.02	0.03	7.07	5.02	10.35	7.81
	vykreslení	0.72	0.64	4.73	4.24	3.18	2.92	4.90	3.74	12.34	10.37
	celkem	1.34	1.22	4.88	4.40	3.20	2.95	11.97	8.76	22.69	18.18

Tabulka 4.4: Srovnání datových typů double (D) a float (F) při použití KD stromů.

přesností bude vždy rychlejší. Je tomu tak pouze v případě, že byla aplikace kompilována na 32-bitovém PC. Při kompilaci pro 64-bitové PC je ale většinou rychlejší `double`.

Verze GCC	**3.4**	**4.1**	**4.2**	**4.3**
`double`	2.51	2.02	1.98	2.18
`float`	2.36	2.41	2.37	2.53

Tabulka 4.5: Výkon po kompilaci v různých verzích překladače GCC.

Příčinou rychlejší práce s čísly s dvojitou přesností jsou optimalizace, které překladač provádí. Tabulka 4.5 obsahuje časy naměřené po kompilaci různými verzemi překladače GCC na scéně ships. Při bližším zkoumání a srovnávání vygenerovaného asembleru jsem zjistil, že `double` verze má kratší kód s daleko menším počtem instrukcí skoků. Zároveň jsou instrukce lépe uspořádány do logických celků, což napomáhá zřetězenému zpracování více instrukcí v procesoru.

Kapitola 5

Závěr

V této bakalářské práci jsem se zabýval zobrazením metodou vykreslení paprsku, především pak optimalizací dělením prostoru pomocí BSP a KD stromů. Testování ukázalo, že pro většinu scén jsou daleko vhodnější KD stromy i přes jejich výpočetně náročnější konstrukci. Kvalitnější konstrukce stromu totiž v naprosté většině případů velice zkracuje čas vykreslování scény.

Práce mě velmi obohatila, podstatně rozšířila mé znalosti v oboru počítačové grafiky a optiky. Také jsem si vyzkoušel objektový návrh a implementaci rozsáhlejšího projektu v jazyce C++ a naučil jsem se tvořit paralelně pracující aplikace pomocí POSIXových vláken a OpenMP.

Do budoucna je aplikace připravena na implementaci různorodých rozšíření. Rád bych ji rozšířil o podporu dalších typů objektů scény (například válec nebo kužel), dalších typů světel a texturování pomocí bitmapových a procedurálních textur. Aplikace může také sloužit jako solidní základ pro implementaci zobrazování metodou *path-tracingu* nebo *photon-mappingu*.

Z hlediska uživatelské přívětivosti by bylo vhodné aplikaci rozšířit o grafické uživatelské rozhraní, nejlépe některou z multiplatformních knihoven jako například wxWidgets, Qt nebo GTK+.

Literatura

[1] WWW stránky. Phong shading. `http://en.wikipedia.org/wiki/Phong_shading`. [Online, přístupné 15.3.2008].

[2] WWW stránky. Realistic raytracing. `http://web.cs.wpi.edu/~emmanuel/courses/cs563/write_ups/zackw/realistic_raytracing.html`. [Online, přístupné 15.3.2008].

[3] Andrew S. Glassner, Jim Arvo, Robert L. Cook, Eric Haines, Pat Hanrahan, Paul Heckbert, and David B. Kirk. *An Introduction to Ray Tracing*. Academic Press, London, 1989. earlier versions as course notes at SIGGRAPH '87 (with Rick Speer) and '88.

[4] WWW stránky. 3d object intersection. `http://www.realtimerendering.com/int/`. [Online, přístupné 10.3.2008].

[5] Tomas Moeller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *journal of graphics tools*, 2(1):21–28, 1997.

[6] WWW stránky. Adaptive sub-sample. `http://www.demoscene.hu/~picard/h7/subsample/subsample.html`. [Online, přístupné 22.3.2008].

[7] Vlastimil Havran. *Heuristic Ray Shooting Algorithms*. PhD thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000.

[8] WWW stránky. Binary space partitioning trees faq. `http://www.faqs.org/faqs/graphics/bsptree-faq/`. [Online, přístupné 10.3.2008].

[9] Jim Arvo. A simple method for box-sphere intersection testing. In *Graphics Gems*, pages 335–339. Academic Press, San Diego, 1990.

[10] Tomas Akenine-Möller. Fast 3d triangle-box overlap testing. *journal of graphics tools*, 6(1):29–33, 2001.

[11] Ingo Wald. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Computer Graphics Group, Saarland University, 2004.

[12] Ingo Wald and Vlastimil Havran. On building fast kd-trees for ray tracing, and on doing that in $o(n \log n)$. pages 61–69, September 2006.

[13] WWW stránky. Distributed ray tracing. http://www.cs.helsinki.fi/group/goa/render/rt/dist/dist.html. [Online, přístupné 15.3.2008].

[14] WWW stránky. Tinyxml. http://www.grinninglizard.com/tinyxml/. [Online, přístupné 22.3.2008].

[15] WWW stránky. Mersenne twister: A random number generator. http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html. [Online, přístupné 22.3.2008].

[16] WWW stránky. Jacco bikker – raytracing: Theory & implementation. http://www.devmaster.net/articles/raytracing_series/part1.php. [Online, přístupné 10.3.2008].

[17] Amy Williams, Steve Barrus, R. Keith Morley, and Peter Shirley. An efficient and robust ray-box intersection algorithm. *journal of graphics tools*, 10(1):49–54, 2005.

[18] WWW stránky. Box-muller transform. http://en.wikipedia.org/wiki/Box-Muller_transform. [Online, přístupné 15.3.2008].

[19] WWW stránky. Openmp tutorial. https://computing.llnl.gov/tutorials/openMP/. [Online, přístupné 10.3.2008].

Dodatek A

Překlad aplikace

Překladový systém aplikace `raytracer` je založen na použití Makefile souborů. Pro správnou funkci je potřeba utility `gmake` ve verzi 3.81 a vyšší. Parametry překladu lze nastavit buď v souboru `config.mk` (resp. `config-win.mk` pro MS Windows verzi), nebo přímým zadáním na příkazové řádce.

A.1 Možnosti překladu

Proměnná	Výchozí hodnota	Popis
`D`	0	Zapnutí ladících výpisů.
`DD`	0	Kompilace s debug informacemi.
`P`	0	Kompilace pro profiler, používejte s `OMP=0`.
`V`	0	Podrobný výpis překladu.
`SDL`	1	Podpora výstupu do SDL okna, vyžaduje knihovnu `libSDL` a `sdl-config`.
`PNG`	1	Podpora výstupu do formátu PNG, vyžaduje knihovnu `libpng`.
`OMP`	1	Použití knihovny `OpenMP`, vyžaduje překladač s podporou (například GCC ve verzi 4.2 a vyšší).
`USE_FLOAT`	0	Použití datového typu `float` pro čísla s plovoucí desetinnou čárkou namísto výchozího typu `double`.

Tabulka A.1: Možnosti překladu aplikace.

A.2 Příklady použití

```
make SDL=0 PNG=0
```

Kompilace bez podpory výstupu do SDL okna a souborů formátu PNG.

```
make USE_FLOAT=1 OMP=0
```

Kompilace s použitím datového typu `float` a bez podpory `OpenMP`.

Dodatek B

Ovládání aplikace

Pokud spustíme aplikaci bez parametrů, případně s parametrem `-h`, uvidíme následující nápovědu:

```
Usage: raytracer [options] filename
Renders a scene specified by filename using ray-tracing.

Rendering options:
  -o output     output type (null png sdl text tga)
  -f filename   filename for text/tga/png output, use - for standard output
  -x width      output width in pixels
  -y height     output height in pixels
  -d depth      recursion depth (for reflections, 1-10)
  -s            enable shadows
  -B            use BSP tree for rendering
  -K            use KD tree for rendering

Image quality options:
  -S samples    enable stochastic super-sampling
  -D option     enable distributed raytracing, possible options:
                s - soft shadows
                d - diffuse reflections
                b - soft shadows + diffuse reflections
  -r samples    samples for distributed raytracing (default 128)

Other options:
  -h            this help
  -v            version info
  -q            quiet mode
  -b            statistics suitable for scripts
  -p            preprocessing only (implies null output)

Raytracer version 1.0 OMP
Copyright (c)2007-2008 Martin Striz <xstriz02@stud.fit.vutbr.cz>
```

Parametrem `-o` zvolíme jednu z možností výstupu vykreslování:

- `null` – výstup naprázdno, používá se pro měření výkonu. Pokud není aplikace kompilována s podporou SDL, tak je tato volba výchozí.
- `sdl` – vykreslení scény do SDL okna. Okno zavřete klávesou `Esc`.
- `png` – výstup do obrázku formátu PNG. Vyžaduje parametr `-f` a knihovnu `libpng`.
- `text` – výstup do textového souboru (ASCII art), vyžaduje parametr `-f`.
- `tga` – výstup do obrázku formátu TGA, vyžaduje parametr `-f`.

Parametry `-x` a `-y` nastavují rozlišení výstupu, `-s` povoluje zpracování stínů a `-d` určuje maximální hloubku rekurze při trasování paprsku. Optimální hodnota pro kvalitní výstup je 5. Dvojice parametrů `-B` a `-K` zapíná použití BSP/KD stromů. Komplexní scény doporučuji vykreslovat s použitím těchto parametrů, jinak bude čas zpracování velice dlouhý. Kvalitu výstupu můžeme zlepšit parametrem `-S`, který zapíná stochastický super-sampling. Hodnota `samples` udává počet paprsků na pixel. Dalším způsobem zvýšení kvality výstupu je použití parametrů `-D` a `-a`, které slouží k povolení a nastavení distribuovaného raytracingu.

Pokud použijeme parametr `-b`, budou statistiky vykreslování vypsány ve formátu, který je vhodný pro zpracování pomocí skriptů, například:

```
0.09 3.51 7383 3 12423 10.05
```

Význam čísel je následující (zleva doprava): čas předzpracování (konstrukce stromu), čas vykreslování, počet objektů, počet světel, počet uzlů BSP/KD stromu, průměrný počet průsečíků s objekty na paprsek.

Parametr `-h` vypisuje nápovědu, `-v` informace o verzi a kompilovaných možnostech výstupu, `-q` potlačuje výstupy aplikace na standardní výstup a `-p` provádí předzpracování scény bez jejího vykreslení.

B.1 Příklady použití

```
raytracer -K -d5 -s -S 25 scenes/balls.xml
```

Vykreslení scény do SDL okna s použitím KD stromu, hloubkou rekurze 5 a zapnutými stíny. Pro zvýšení kvality výstupu je použit super-sampling, který pracuje s 25 paprsky na každém pixelu.

```
raytracer -B -o tga -f out.tga scenes/ships.xml
```

Výstup vykreslování do souboru `out.tga` ve formátu TGA s použitím BSP stromu.

Dodatek C

Obsah CD

Na přiloženém CD se nacházejí tyto soubory a adresáře:

- soubor `thesis.pdf` – technická zpráva ve formátu PDF
- adresář `app` – zdrojové texty aplikace, ukázkové scény a modely
- adresář `doc` – manuálová stránka `raytracer.1` a programová dokumentace vygenerovaná pomocí aplikace Doxygen. Titulní stránka je v souboru `index.html`
- adresář `poster` – plakát k bakalářské práci
- adresář `thesis` – zdrojové texty technické zprávy (LaTeX).

Dodatek D

Specifikace RAW formátu

Každý soubor s 3D modelem je uvozen identifikačním řetězcem `raw3d`, po něm následuje jeden z těchto znaků:

- `T` – model je složen z trojúhelníků
- `S` – model je složen z koulí

Dále následuje jeden 32-bitový `unsigned int` s počtem primitiv. Za ním následují data modelu. Pro modely složené z trojúhelníků jsou to tři čísla typu `float`, které udávají souřadnice vrcholu. Tři po sobě jdoucí vrcholy tvoří trojúhelník. Model složený z koulí obsahuje čtveřice čísel datového typu `float`: první tři určují souřadnice středu, čtvrté číslo poloměr.

`raw3d`	typ modelu (`T`/`S`)	počet objektů podle typu

Tabulka D.1: Hlavička RAW modelu